Performance evaluation of block acquisition and tracking algorithms using an open source GPS receiver platform

Ganesh K. Ramachandran, David Akopian
The University of Texas at San Antonio
Gregory W. Heckler, Luke B. Winternitz
NASA-Goddard Space Flight Center

BIOGRAPHY

Ganesh Kumar Ramachandran received his M.S degree in Electrical Engineering from The University of Texas at San Antonio, in 2010 and B.E. degree in Electrical and Electronics Engineering from Anna University, India, in 2006. Prior to joining UTSA he was a Junior Test Engineer at M/s Quintegra Solutions Ltd for 2.5 years. His current research interests include real-time software GPS receiver implementation on various open source platforms and receiver testing modes.

David Akopian is an Associate Professor at the University of Texas at San Antonio (UTSA). He joined the UTSA in 2003 where he founded Software Communication and Navigation Systems Laboratory. He received the M.Sc. degree in radio-electronics from the Moscow Institute of Physics and Technology in 1987 and Ph.D. degree in electrical engineering from the Tampere University of Technology (TUT), Finland, in 1997. From 1999 to 2003 he was a Senior Engineer and Specialist with Nokia Corporation. Prior to joining Nokia in 1999 he was a member of teaching and research staff of TUT. His current research interests include digital signal processing algorithms for communication and navigation receivers, positioning methods and mobile applications.

Gregory W. Heckler is an aerospace engineer in the Communications Systems Branch (Code 567) at NASA Goddard Space Flight Center. He holds a B.S. and M.S. degree in Aerospace Engineering from Purdue University. His research interests include GPS space applications and software radio.

Luke B. Winternitz is a an electrical engineer in the Component and Hardware Systems Branch (Code 596) at NASA Goddard Space Flight Center in Greenbelt, Maryland. He has worked at Goddard for nine years primarily in the development of GPS receiver technology. He received Ph.D. degree in electrical engineering from the University of Maryland, College Park, in 2010. His research interests include GPS space applications and software radio.

ABSTRACT

Location technologies have many applications in wireless communications, military and space missions, etc. US Global Positioning System (GPS) and other existing and emerging Global Navigation Satellite Systems (GNSS) are expected to provide accurate location information to enable such applications. While GNSS systems perform very well in strong signal conditions, their operation in many urban, indoor, and space applications is not robust or even impossible due to weak signals and strong distortions. The search for less costly, faster and more sensitive receivers is still in progress.

As the research community addresses more and more complicated phenomena there exists a demand on flexible multimode reference receivers, associated SDKs, and development platforms which may accelerate and facilitate the research. One of such concepts is the software GPS/GNSS receiver (GPS SDR) which permits a facilitated access to algorithmic libraries and a possibility to integrate more advanced algorithms without hardware and essential software updates. The GNU-SDR and GPS-SDR open source receiver platforms are such popular examples.

This paper evaluates the performance of recently proposed block-correlator techniques for acquisition and tracking of GPS signals using open source GPS-SDR platform.

I. INTRODUCTION

In conventional GPS [1],[2], satellites orbiting the earth transmit direct sequence spread spectrum (DSSS) ranging signal on the carrier frequencies of L1 (1575.42 MHz) and L2 (1227.60 MHz).

The GPS receiver measures the time-of-transmission (TOT) of the satellite ranging code. Based on this and its own time, it determines the time required for the signal to propagate from the satellite to the receiver. This time is converted to a distance when it is multiplied by the speed of light. The receiver also decodes the navigation data from satellite messages, which contains orbital parameters, correction data, time stamps, etc. Using navigation data and transmission times the receiver estimates the locations of GPS satellites. Having the satellite locations, and satellite-to-receiver distances, one can unambiguously determine the position of the receiver using trilateration techniques. The estimation can also resolve for the receiver clock error as it is generally

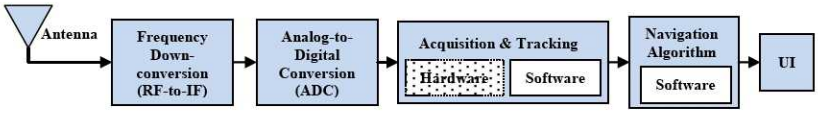


Figure 1. A schematic structure of a GPS receiver. Hardware accelerators are often completely excluded in software defined GPS

biased. Least Squares (LS) or closed form methods can be used to solve GPS trilateration equations.

In the conventional civilian receiver each satellite modulates the sinusoidal L1 carrier signal with a unique coarse acquisition (C/A) code known at the receivers. The C/A code is a binary pseudorandom noise (PRN) code sequence comprising 1023 chips repeating each millisecond. The C/A code is 1ms long containing values of -1 and +1.

The L1 signal is further modulated with the navigation data at a bit rate of 50 bit/s. Thus, the transmitted signal is a product of a data component, a PRN code component and a sinusoidal carrier component. The primary measurement tasks at the receiver include calculation of a range (satellite-to-receiver distance), range rate, and demodulation of the navigation data. For range and range-rate measurements the receiver has to synchronize locally generated 'replica' code with the received signal in order to de-spread the code and estimate delays. The synchronization is done in two phases; coarse synchronization (acquisition) and fine synchronization (tracking).

In both modes, correlators are used to find the best alignment of the received signal and replica code sequence and thus to find their relative signal shift (delay) called 'code-phase'. The notion of the code-phase is due to the DSSS signal structure, which has the same pseudorandom signal pattern periodically repeating in time. The signals are aligned when the edges of the code periods are aligned. Figure 1 illustrates the structure of a conventional GPS receiver. Acquisition and tracking modules may reuse correlators, and most of the state-of-the-art receivers use dedicated hardware accelerators for the correlators. With the development of faster processing units and more reconfiguration capability needs the correlators can be completely implemented in software, a concept known as software defined radio or receiver (SDR) [8]-[11].

Dedicated hardware has clear advantages in faster processing but the functionality is limited to the particular design of each chip. Software implementations are becoming more and more attractive due to their flexibility to adapt to GPS signal modernization, new Global Navigation Satellite System (GNSS) signals [2], and demand of multimode weak signal processing algorithms [2],[3] to handle many possible scenarios and multi-sensor integrations. Examples of software GPS receiver implementations are [4]-[8]. Tight integration of baseband and positioning algorithms of the GPS receiver along with other sensors is a big advantage of software receivers. Software implementations may reduce the cost of the positioning systems as new versions can be simply installed as software upgrades.

To make use of the advantages and potential of software implementations, computationally efficient algorithms should be used to implement massive correlators employed in the state-of-the-art receivers for high sensitivity. This is a very big challenge for software implementations. GPS receivers process signals in three dimensional uncertainty space: 'available satellites', 'code-phase of each satellite', and 'Doppler frequency shifts'. Doppler shifts are due to relative satellite-receiver movement and clock inaccuracies. The Doppler shift causes changes both in code rate and carrier frequency. For example, in terrestrial applications, the maximum change in the apparent received carrier frequency due to Doppler is estimated to be less between ±10kHz.

This multidimensional search requires significant computational resources. In addition, unlike conventional software communication systems, positioning receivers deal with weak signals and need long integration times (on the order of seconds) to detect signals in difficult environments. Thus, new algorithms with significantly reduced computational complexities are required for "software" implementations.

Recently, block-correlator algorithms have been suggested for drastic computational reductions both in the frequency [12], [13] and time domain [14]. In these algorithms, many operations are shared and computational redundancy is minimized, resulting in significantly faster processing. Due to computational architecture specifics, the arithmetic operations reduction may not necessarily result in faster processing, as it depends on many factors related to data flow. This paper describes implementations of two block-correlator concepts for acquisition and tracking on an open source platform to validate theoretical performance acceleration estimates on a real-time GPS receiver platform. It is demonstrated that indeed this performance improvement is achieved as compared to alternative methods.

Comment [LBW21]: Just needs some summary of the positive results achieved.

The paper is organized as follows. Section II concisely presents the GPS-SDR open source software receiver platform. Section III describes fast FFT-based block-correlator for signal acquisition. Section IV provides a

time-domain block-correlator for tracking. Algorithm performance results for GPS-SDR receiver are shown in Section V and conclusions are made in Section VI.

II. GPS-SDR OPEN SOURCE PLATFORM

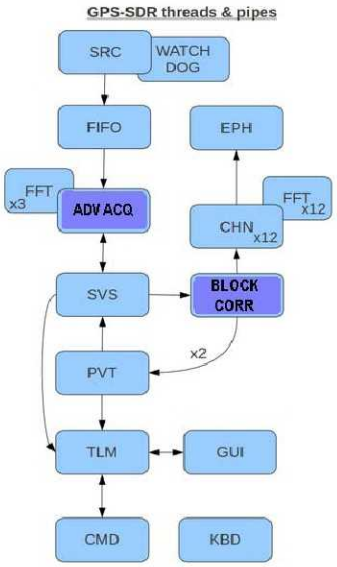


Figure 2. New GPS-SDR Architecture [8].

The GPS-SDR [8] open source project is a popular real-time C/C++-based GPS receiver which can be used to evaluate the overall performance of the block-correlator algorithms by replacing appropriate blocks. The GPS-SDR is compatible with popular RF front-ends, USRP [11] and SiGe [4]. Real-time signal processing is achieved through the use of carefully coded low-level processing routines. The GPS-SDR is highly modular and multithreaded enabled. Currently it processes GPS L1 C/A signals, and can handle certain weak signal conditions. This receiver runs on a PC/laptop which connects to the front-end through a USB 2.0 port. Other extensions such as L2 signal processing are being implemented. The receiver has a built-in GUI that allows the user to conveniently interact with the receiver during the operation. This receiver also has advanced capabilities to determine the velocity and time information during motion. The acquisition unit of the receiver is designed and developed for both strong and weak signal acquisition with the capability for both coherent and non-coherent integration over different GPS signal durations of GPS signal. For faster processing, the original correlations are performed using assembly level functions to meet critical processing speeds for real-time receiver operation. Figure 2 illustrate the architecture of the modified GPS-SDR project with incorporated new advanced acquisition and tracking modules highlighted. While many components of GPS-SDR employ conventional approaches, novel technical solutions such as optimized FIFO make the real-time processing feasible.

III. FAST BLOCK-CORRELATOR ALGORITHM FOR ACQUISITION

The acquisition is the first step of the synchronization and it is typically the most computationally intensive stage as compared to tracking. Block correlators algorithms are proposed to reduce arithmetic complexity in [12],[13]. Both approaches employ two stage Doppler frequency compensations: (1) coarse frequency compensation in integer kHz steps and (2) sub-kHz fine frequency compensation. The GPS-SDR platform originally implemented the approach in [13] in which fine Doppler frequency shift compensation is implemented after the correlators.. This results in correlation peak degradations

Comment [LBW22]: Should define all acronyms when first used.

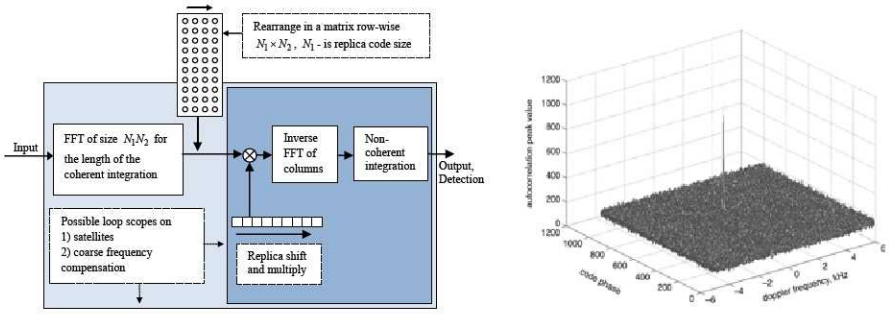
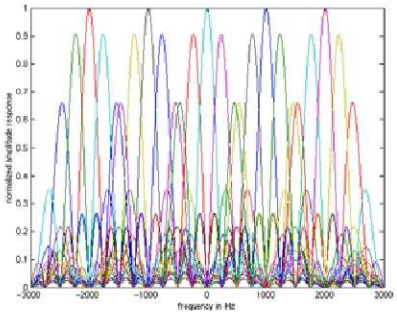
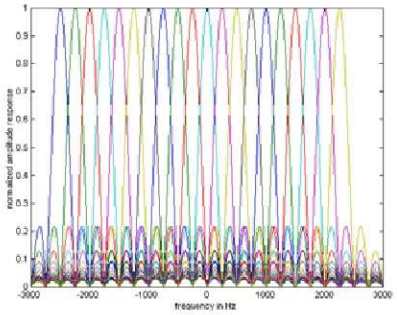


Figure 3. (Left) Block-diagram of Fast FFT based Acquisition [12]; (Right) Correlation peak using of Fast FFT based acquisition technique

(see Figure 4a). As a result, the coarse frequency spacing needs to be more dense, e.g., about 250kHz or 500kHz to be able to acquire weak signals. This paper employs the method introduced in [12] (Figure 3) as it is free from peak degradation phenomena (Figure 4b).



(a)



(b)

Figure 4. Examples of correlation peak degradations due to residual Doppler frequency shifts for the acquisition with integer kHz coarse Doppler frequency compensation. (a) after fine frequency compensation by the method in [13]; (b) after fine frequency compensation by the method in [12].

Computational savings are achieved through block-processing by optimizing the coherent processing stage. Initially, a coarse Doppler frequency compensation is performed with frequencies as integer kHz. This can be performed by multiplying the input with a sinusoid at a compensating frequency (e.g. p kHz, $p=-10,-9,...,9,10$), or equivalently by cyclically shifting the FFT of the replica signal by p samples. Sub-kHz Doppler frequencies are processed jointly. Let input samples x_n be arranged in a matrix $\mathbf{X}$ filled column-by-column. In other words, the matrix $\mathbf{X}$ contains elements $x_{n_1,n_2}=x_{n_1+n_2N_1}$, where $n_1=0,...,N_1-1$ and $n_2=0,...,N_2-1$, N_1 is the code period length in samples, and N_2 is the number of coherently combined code periods. The frequency resolution will be $1/N_2$ kHz. Combining multiple code periods of the input signal coherently with Doppler compensation can be performed as:

$$\mathbf{Z}^{coh}=\mathbf{C}.*\left(\mathbf{X}\mathbf{F}^T\right) \quad (1)$$

where $\mathbf{F}$ is the DFT matrix, notation $.*$ denotes element-wise multiplication, and $\mathbf{C}$ has elements

Comment [LBW23]: Define this

$$C_{n_1,k}=e^{-2\pi j\cdot\frac{kn_1}{N_1N_2}} \quad (2)$$

Columns of the matrix $\mathbf{Z}^{coh}$ are coherently combined epochs with a candidate Doppler frequency wiped-off. Each column corresponds to a certain frequency from the group of frequencies defined by the index k: $f_k=f_s\frac{k}{N_1N_2}$, where $k=0,1,...,N_2-1$, and f_s is the sampling frequency. Equation (1) states that all the frequencies corresponding to the values of index $k=0,1,...,N_2-1$ are processed jointly using the DFT. Each of the columns of the matrix $\mathbf{Z}^{coh}$ is then correlated with the replica code at all possible code phases. In the frequency domain, using the convolution theorem, one can obtain for the output of the coherent stage

$$\mathbf{Z}^{corr}=\mathbf{F}_{N_1}^{-1}\mathbf{R}_f\mathbf{F}_{N_1}\mathbf{C}.*\left(\tilde{\mathbf{X}}\mathbf{F}_{N_2}^T\right) \quad (3)$$

Here, $\mathbf{R}_f$ is a diagonal matrix, $diag\left(\mathbf{R}_f\right)=\mathbf{F}_{N_1}\mathbf{r}$, $\mathbf{r}$ is the inverted replica code epoch with zero code phase, $\mathbf{F}_N$ is the DFT matrix of size N. It can be shown that in (3), the fragment $\mathbf{F}_{N_1}\mathbf{C}.*\left(\tilde{\mathbf{X}}\mathbf{F}_{N_2}^T\right)$ can be implemented by a single DFT (implemented using the FFT algorithm). Thus one FFT is used for a joint processing of multiple Doppler frequencies and code-phases. The overall processing structure is shown in Figure 3.

IV. A FAST BLOCK-CORRELATOR ALGORITHM FOR TRACKING

The tracking loop follows the incoming signal and adjusts itself to de-spread and de-modulate the incoming signal. Two tracking loops are used to track the incoming GPS signal: a delay-locked loop (DLL) to track the code and a frequency (FLL) and/or phase locked loop (PLL) to track the frequency/phase of the incoming signal. See Figure 5. Here, the DLL consists of early, prompt, and late code generators, filters and discriminators. The early and late codes are half a chip (or less) time shifted versions of the prompt code. The incoming signal is correlated with early

and late C/A codes to produce two outputs which are fed to a discriminator. A control signal is generated based on discriminator's output to adjust the rate of the locally generated C/A code to match the C/A code of the incoming signal. Different discriminators can be used. The navigation data is finally extracted by de-spreading the received GPS signal with the locally generated prompt code. Advanced DLL tracking loops may use more correlators to address multipath and non-triangular autocorrelation shapes.

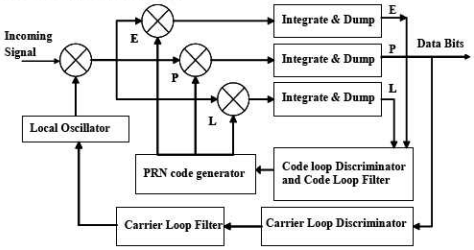


Figure 5. Block diagram of the combined tracking loop.

As for the frequency tracking; phase and/or frequency locked loop (PLL/FLL) can be used [1],[2]. Conventional PLL loops generate a local carrier signal that is driven to alignment with the incoming signal.. In the block correlation, approach there is no need to generate the local carrier for every incoming sample. Sine and cosine tables are generated and stored in memory once, and using the feedback from the PLL loop, the local sinusoid is designed to run either slow or fast [15]. In the GPS-SDR, a Costas Loop is used (Figure 6) for the implementation of a PLL feedback loop. Costas loops are insensitive to 180° phase transitions due to navigation data bits and they are typically preferred choice in GPS receivers.

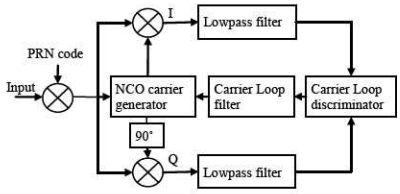


Figure 6. Costas PLL for carrier tracking

In the tracking block-correlator all three correlations are computed together. As opposed to conventional correlators, block correlators use fixed groups of replica sequences. The errors from the tracking loop just switches replica selections from one group to another when needed.

The joint block correlation is described next following the original approach in [14]. For performing correlation operations, samples of the received signal are stored in memory. This set of samples is denoted as: $x_{N-1}, x_{N-2}, ..., x_3, x_2, x_1, x_0$ where N is the number of samples. In this paper we compute K correlations in block processing mode. The replica code sequence is set as $r_{N-1}^k, r_{N-2}^k, ..., r_3^k, r_2^k, r_1^k, r_0^k$, where $k \in \{1, ..., K\}$ identifies the respective replica code sequence and $r_n^k \in \{-1, +1\}$. Denoting the consecutive correlation values as C^k, the operation of conventional correlators is defined as:

$$C^k = \sum_{j=0}^{N-1} r_j^k x_j \ , \ k \in \{1, ..., K\} \ . \qquad (4)$$

For the block processing method (Figure 7), the received samples which multiply the same set of replica samples in a group of correlators are grouped together. Let the index variable j identify received samples and $J_{b_1 ... b_K}$ be the set of these indices belonging to the same group. Formally $j \in J_{b_1 ... b_K}$ if $\left(r_j^1, ..., r_j^K\right) = \left(b_1, ..., b_K\right)$, $b_i \in \{-1, +1\}$. There are 2^K such groups. Then (4) becomes

$$\begin{aligned} C^k &= \sum_{\substack{\text{All } 2^K \\ \text{combinations} \\ b_1, ..., b_K}} \ \sum_{j \in J_{b_1 ... b_K}} r_j^k x_j \\ &= \sum_{\substack{\text{All } 2^K \\ \text{combinations} \\ b_1, ..., b_K}} b_k \sum_{j \in J_{b_1 ... b_K}} x_j = \sum_{\substack{\text{All } 2^K \\ \text{combinations} \\ b_1, ..., b_K}} b_k S_{b_1 ... b_K} \end{aligned} \qquad (5)$$

The fast block processing algorithm is based on the idea that samples from each group will be used only once for computing sub-sums $S_{b_1 ... b_K} = \sum_{j \in J_{b_1 ... b_K}} x_j$. These sub-sums are then used for computing K correlations. The number of additions is reduced almost K times if the number of sub-sums is significantly less than the number of samples. For $K = 3$ there are eight groups $J_{-1,-1,-1}$ to $J_{+1,+1,+1}$, each identifying a group of samples. A register is assigned to each group to store sub-sum $S_{b_1 ... b_K}$. All such registers are also indexed as $\left(b_1 .. b_K\right)$. For illustration purposes our figures also use an equivalent notation for $\left(b_1 .. b_K\right)$ where signed binary values of b_k are replaced with (0, 1) binary values or $\left(b_1 .. b_K\right)$ is replaced by an integer. Example: $(+1, -1, +1) \rightarrow (1, 0, 1) \rightarrow 5$. Here $(1, 0, 1)$ is the binary codeword of 5.

For calculating the sub-sum $S_{b_1 ... b_K}$ for each group and for each of the correlator iterations, all 2^K registers are initialized to zero. Then, the algorithm processes the stored received samples x_j one after the other forming the sub-sums. In Figure 7a, one of the received samples is

Comment [LBW24]: Maybe a reference here.

denoted as x_n. The samples of the three replica code sequences having the same index n are $r_n^1 = -1$, $r_n^2 = +1$, $r_n^3 = -1$. These samples correspond to the register address (010) or "2". The adder adds the received value x_n to the sub-sum S_2 and stores the new value of sub-sum S_2 into the register again. This procedure is performed analogously for all N received samples. The combining unit of Figure 7a then combines all stored sub-sums $S_{b_1 \ldots b_K}$ to obtain the K correlation values C^k at once as a result of block processing. Note that multiplications $b_k S_{b_1 \ldots b_K}$ are just sign changes as b_k is either $+1$ or -1. Figure 7a illustrates an example of how the sub-sums are formed while Figure 7b presents an example how the sub-sums are combined to produce three parallel outputs corresponding to the correlations with three parallel replicas. In the example of Figure 7, correlation values C^1, C^2, C^3 have to be calculated for $K = 3$ replica code sequences, and the number of groups is eight. The complexity reduction estimate is about 3 times for three joint correlations. The integration of the block correlator into the tracking loop is illustrated in Figure 8 [15].

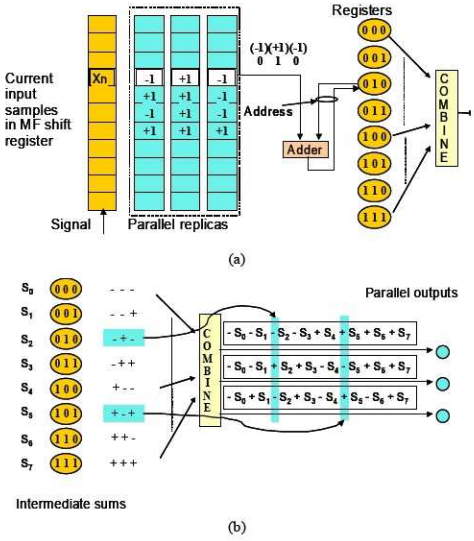


Figure 7. (a) Suggested block correlator structure. An example with three replica sequences; (b) sub-sum combining [14],[15]

The block-correlator described above has been integrated into the GPS-SDR. Once satellites are acquired, the initial estimates of code phases and Doppler frequencies of each satellite are provided to the tracking channels dedicated to individual satellites. The tracking loops continuously track the variations in received frequency and code phase due to the line of sight (LOS) dynamics between the GPS satellites and the receiver.

To implement the code-phase corrections, the GPS-SDR's original implementation has several versions of replicas corresponding to different code phases. Depending on the DLL output error, the appropriate replica is selected and multiplied against the next 1ms signal fragment. The next fragment is chosen according to the code phase error obtained after tracking the previous 1 ms duration of the GPS signal. To validate the block correlation algorithm with the GPS-SDR, binary indices described in the algorithm itself are created for all the versions of original replicas. Later, the binary index values for a particular replica version is chosen for tracking using block correlation algorithm according to the code phase error.

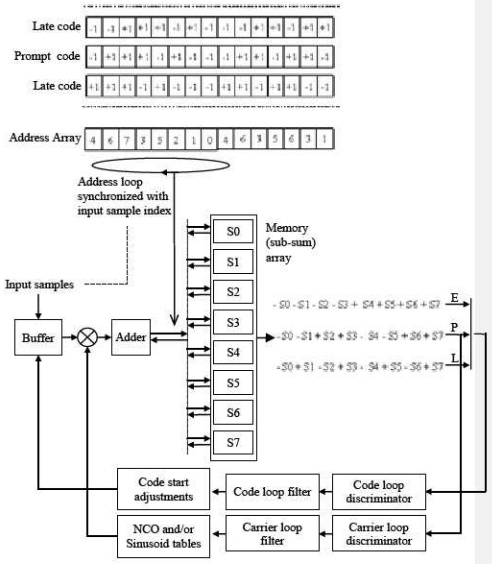


Figure 8. Block correlator structure used in the combined tracking loop [14],[15]

V. PERFORMANCE EVALUATION

The block correlator requires only additions since all the multiplications in the equations can be implemented with sign changes. The computational performance of block-correlators for acquisition and tracking are evaluated in the GPS-SDR. The GPS-SDR receiver is implemented in C++ with some lower level functions such as the FFT and 1ms correlations implemented in assembly code. The same assembly level FFT is used for the acquisition block correlator.

The performance of the algorithms described in this paper have been estimated using the GPS-SDR running on a PC with the following parameters: Intel Core 2 Duo Processor, 2.00GHz clock, 2GB RAM, LINUX-UBUNTU OS. 'OProfile' profiling tool is used for CPU load estimations [16]. 'Oprofile' is a system-wide profiler for Linux systems with a capability of profiling all codes with little overhead [16]. 'Oprofile' is released under the GNU GPL, and sample data are collected using a kernel driver and a daemon. The collected data are later converted to useful information using several post-profiling tools of 'Oprofiler'. 'Opconrol' and 'Opreport' scripts are mainly used to profile the C++ codes using 'Oprofiler'. The 'Opcontrol' script is run to: start profiling, end a profiling session, dump profile data, and set up the profiling parameters. The 'Opreport' script is used to output binary image summaries, or per-symbol data from 'Oprofile' profiling sessions, which includes the CPU percentage timings of processes running in the PC.

Table1. 'Oprofiler'-based 'CPU Load' estimations for acquisition

	Block-correlator [12]	Conventional Block-correlator [13]
CPU Load (%) of acquisition as a fraction of overall GPS_SDR load	30.6%	17%

Table2. 'Oprofiler' CPU average load estimations for tracking

	Block-correlator [14]	Conventional Correlator, Assembly	Conventional Correlator, C++
CPU Load	11%	17%	33.79%

The CPU loadrequired by the implemented block correlator [12] is compared with original GPS-SDR block-correlator implementation [13] using the 'Opcontrol' and 'Opreport' command scripts of the profiler. In addition, clocking commands such as 'clockbeg()' and 'clockend()' are used in the receiver code to determine the overall time consumption.

The GPS-SDR's original acquisition algorithm was run for the whole set of 32 satellite codes. Coherent integration length was set to 8ms. The execution time estimates using clocking tools were found to be 0.6 second for [12] and 5.32 seconds for [13]. This performance improvement is partially explained by the extra correlation computations to avoid peak degradations for post-correlation fine Doppler frequency compensation (see Fugure 4a) employed in [13]. For this experiment the original GPS-SDR performs four iterations of 'coarse' acquisition with 250kHz Doppler frequency shifts for each 1kHz frequency search range. Even with four iterations, the block-correlator from [12] takes only 4 seconds. Table 1 shows the average CPU load estimated by the profiler in offline mode for the duration of acquisition. One can apparently see the efficiency of the block correlator from [12].

The performance evaluation of the tracking algorithms was performed on captured data (up to 30 seconds) for 12 channels on the same PC. For 30 seconds of recorded signal, the computation time of the block-correlator approach [14] was found to be 13.25 seconds compared to the original GPS-SDR's conventional correlation performed in assembly level, which was about 19 seconds. The conventional correlation performed in C++ language was about 23 seconds. Figure 10 demonstrates the performances for various signal durations. Table 2 shows CPU loads by the block-correlator approach (about 11%), the conventional correlation implemented in C++ (about 33.79%) and GPS-SDR assembly code implementations (about 17%).

VI. CONCLUSIONS

This paper describes the implementation of two recently proposed block-correlators for acquisition and tracking on the open source GPS receiver platform GPS-SDR. For acquisition, computational gains are achieved using special FFT based processing. For tracking, a time-domain block correlator exploits joint processing of early, prompt and late correlations. Performance benchmarking results demonstrate significant improvement over conventional tracking correlators and an alternative block-correlator acquisition algorithm.

Comment [LBW25]: In tables 1 and 2 be consistent with number of significant figures

Comment [LBW26]: More comments: Need reference to figure 9 somewhere in the paper. Need units on axes of figures 9 and 10.

REFERENCES

[1] E.D. Kaplan. *Understanding GPS: Principles and Applications*. Boston: Artech House, 1996.

[2] P. Misra, P. Enge. *Global Positioning System, Signals, Measurements, and Performance*. Ganga-Jamuna Press, Lincoln, MA. 2001.

[3] N. Agarwal et al, "Algorithms for GPS operation indoors and downtown," GPS Solutions. 2002, vol. 6, no. 3, pp. 149-160, Springler-Verlag Heidelberg.

[4] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S.H. Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*. Birkhauser Boston. 2006.

[5] D. Akos and M. Braasch, "A Software Radio Approach to Global Navigation Satellite System Receiver Design," 1996 Institute of Navigation Annual Meeting, Cambridge, MA, June 1996.

[6] Software receiver solution from SiRF and Nordnav, currently with CSR. www.csr.com (Access: Dec 20, 2010).

[7] Software Receiver solution from Fastrax. www.fastrax.com (Access: Dec 20, 2010)

[8] Open source GPS software radio - GPS-SDR, multithreaded enabled C++ application. www.gps-sdr.com/wiki/gps-sdr (Access: Dec 20, 2010).
[9] Open source GNU-Radio. www.gnuradio.org. (Access: Dec 20, 2010).
[10] Eric Blossom, "Exploring GNU Radio.", www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html#block-diagram. (Access:Dec 20, 2010).
[11] USRP RF Front-End for software radio. Ettus Research. www.ettus.com. (Access: Dec 20, 2010)
[12] D. Akopian, "Fast FFT based GPS satellite acquisition methods," Proceedings of IEE, Vol. 152, No. 4, pp. 277-286, 2005. Initial version presented at ION-GPS-2001 Conference, Sep. 11-14, 2001, Salt Lake City, UT.
[13] M.L. Psiaki, "Block acquisition of weak GPS signals in a software receiver," Proc. Of ION-GPS-2001 Conference, Sep. 11-14, 2001, Salt Lake City, UT, pp. 2838-2850
[14] D. Akopian, S. Agaian, "Fast and parallel matched filters in time domain," Proc. of ION GNSS 2004 Conference, Sep 21-24, Long Beach, CA.
[15] P. Sagiraju, P. Kashyap, D. Akopian, "Block Correlator for Tracking GPS/GNSS Signals", Proc. of ION GNSS 2008 Conference, Sep 21-24, Savannah, GA.
[16] Oprofile Tool. http://oprofile.sourceforge.net/about/. (Access: Dec 27, 2010).

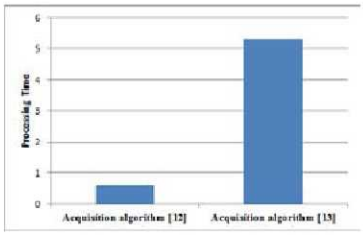


Figure 9. Performance evaluation of block-correlator acquisition algorithms [12] (left) and [13] (right) using clocking tools

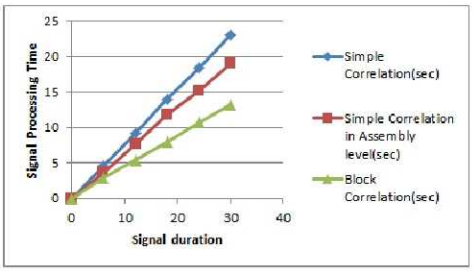


Figure 10. Correlator performance using clocking tools for tracking block-correlator [14] vs conventional correlators implemented in C++ and assembly codes for various signal durations.